Neural cryptography with queries

Andreas Ruttor[1], **Wolfgang Kinzel**[1] **and Ido Kanter**[2]

[1] Institut für Theoretische Physik und Astrophysik, Universität Würzburg, Am Hubland, 97074 Würzburg, Germany

[2] Department of Physics, Bar Ilan University, Ramat Gan 52900, Israel

E-mail: andreas.ruttor@physik.uni-wuerzburg.de, wolfgang.kinzel@physik.uni-wuerzburg.de and kanter@mail.biu.ac.il

Abstract. Neural cryptography is based on synchronization of tree parity machines by mutual learning. We extend previous key-exchange protocols by replacing random inputs with queries depending on the current state of the neural networks. The probability of a successful attack is calculated for different model parameters using numerical simulations. The results show that queries restore the security against cooperating attackers. The success probability can be reduced without increasing the average synchronization time.

Keywords: stochastic processes (theory), neuronal networks (theory), new applications of statistical mechanics

1. Introduction

Neural cryptography [1, 2] is a method for generating secret information over a public channel. Before two partners A and B can exchange a secret message over a public communication channel they have to agree on a secret encryption key. Using number theoretic methods, such keys can be constructed over public channels without previous secret agreements of the two partners [3]. Any details of the algorithm as well as the complete information passed between the partners are known to a possible attacker E; nevertheless the final key is secret, it is known to the two partners A and B only, and an opponent E with limited computer power cannot calculate the key. The method is based on the computational difficulty of factorizing large numbers or calculating the discrete logarithm of large numbers [3].

Recently it has been demonstrated that secret keys can be generated by a completely different method [1], as well. This method is based on synchronization of neural networks by mutual learning [4, 5]. The secret key is generated by the dynamics of a complex physical process, namely the competition between stochastic attractive and repulsive forces which act on the weights of the two neural networks of the partners A and B. Two dynamical systems which synchronize by mutual signals have an advantage over an attacker E which can only synchronize by listening to the exchanged signals [2]. Finally the key is taken as the synchronized weights of the two networks A and B.

The security of neural cryptography is still being debated [6–11]. Since the method is based on a stochastic process, there is a small chance that an attacker synchronizes to the key, as well. However, it has been found that the model parameter L (the synaptic depth defined below) determines the security of the system: the success

probability of the attacker decreases exponentially with L while the synchronization time, i.e. the amount of effort for agreeing on a key, increases by L^2, only. Hence, by increasing the value of L the security of neural cryptography can be increased to any desired level [9].

These scaling laws are not obvious at all. Neural cryptography is based on a subtle difference between bi-directional and uni-directional couplings of a stochastic process. Our understanding of these processes is still incomplete. Hence it is still possible that a clever algorithm may destroy the security of the method.

In fact, there is a special algorithm of neural cryptography, the Hebbian rule defined below, which allows a clear determination of scaling laws with respect to the parameter L. Recently it has been shown that a special attack based on the majority of an ensemble of attacking networks destroys the security of the method: the success probability is constant for large values of L [11]. These results are the motivation for the present investigation. We develop a new kind of learning rule which restores the security: the success probability decreases exponentially with L. This new rule uses *learning by queries* [12] which is a well-known principle in the theory of learning by examples [13]. It is based on exchanging inputs between A and B which are correlated to the weight vectors of the two networks.

It should be mentioned that there are a few other rules (anti-Hebbian or random walk) with lower success probabilities for which the scaling properties with respect to the parameter L cannot be found, yet. This means that we still do not know whether the majority attack destroys the security of those algorithms, as well. Nevertheless it is important to develop a rule which restores security for the Hebbian rule where the majority attack is clearly successful.

Our findings allow us to reach the conclusion that for all algorithms suggested so far the scaling laws for the success probability hold: the security of neural cryptography can be increased to any desired level.

2. Neural cryptography

In this section we repeat the definition of neural cryptography. Each of the two partners A and B uses a special neural network called tree parity machine (TPM). As shown in figure 1, a TPM consists of K hidden units σ_k with weight vectors $\boldsymbol{w}_k$ and input vectors $\boldsymbol{x}_k$. The components of the input vectors are binary and the weights are discrete numbers with depths L,

$$x_{k,j} \in \{-1, +1\}, \quad w_{k,j} \in \{-L, -L+1, ..., L-1, L\}, \tag{1}$$

where the index $j = 1, \ldots, N$ denotes the elements of each vector and $k = 1, \ldots, K$ the hidden units. The outputs of these neurons are defined by the scalar product of inputs and weights,

$$\sigma_k = \mathrm{sgn}(\boldsymbol{w}_k \cdot \boldsymbol{x}_k)\,. \tag{2}$$

The final output bit of each TPM is defined by the product of the hidden units,

$$\tau = \prod_{k=1}^{K} \sigma_k\,. \tag{3}$$

Both partners A and B initialize their weight vectors by means of random numbers before the training period starts. At each time step t a public input vector is generated and the bits τ^{A} and τ^{B} are exchanged over the public channel. In the case of identical

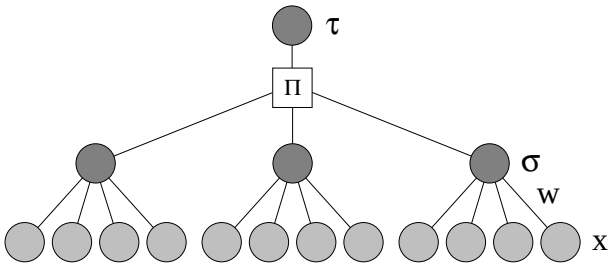

Figure 1. A tree parity machine with $K = 3$ and $N = 4$.

output bits, $\tau^{\mathrm{A}} = \tau^{\mathrm{B}}$, each TPM adjusts those of its weights for which the hidden unit is identical to the output, $\sigma_k^{\mathrm{A/B}} = \tau^{\mathrm{A/B}}$. These weights are adjusted according to a given learning rule. Here we consider the Hebbian rule

$$\boldsymbol{w}_k^{\mathrm{A/B}}(t+1) = \boldsymbol{w}_k^{\mathrm{A/B}}(t) + \tau^{\mathrm{A/B}}\boldsymbol{x}_k\,. \tag{4}$$

After some time t_{sync} the two partners are synchronized, $\boldsymbol{w}_k^{\mathrm{A}}(t) = \boldsymbol{w}_k^{\mathrm{B}}(t)$, and the communication is stopped. Then the common weight vector is used as a key to encrypt secret messages.

Note that any possible attacker E knows as much about the process as A knows about B and vice versa. But E has some disadvantage with respect to A and B: it can only listen to the communication and cannot influence the dynamics of the weights in A's and B's neural networks [2, 14]. It turns out that this difference determines the security of the crypto-system.

It seems obvious that a successful attacker should also use a TPM with similar training rules. In fact, it is advantageous to use many networks for an attack. Since the method is stochastic (due to the random input) the attacker may synchronize as well, but with a low probability P_{E}. The security of neural cryptography is related to the fact that P_{E} decreases exponentially with the value of L [9].

In all previous learning rules the sequence of input vectors $\boldsymbol{x}_k(t)$ was generated by a public random number generator. Here we propose to take queries, i.e. input vectors which are correlated with the present weight vector $\boldsymbol{w}_k(t)$. At odd (even) time steps the partner A (B) is generating an input vector which has a certain overlap to its weights $\boldsymbol{w}_k^{\mathrm{A}}$ ($\boldsymbol{w}_k^{\mathrm{B}}$). It turns out that queries improve the security of the system.

3. Queries

The process of synchronization itself can be described by standard order parameters which are also used for the analysis of on-line learning [13]. These order parameters are

$$Q_k^m \;\; = \frac{1}{N}\boldsymbol{w}_k^m \cdot \boldsymbol{w}_k^m\,, \tag{5}$$

$$R_k^{m,n} = \frac{1}{N}\boldsymbol{w}_k^m \cdot \boldsymbol{w}_k^n\,, \tag{6}$$

where the indices $m, n \in \{A, B, E\}$ denote A's, B's, or E's TPM. The distance between two corresponding hidden units is defined by the (normalized) overlap

$$\rho_k^{m,n} = \frac{\boldsymbol{w}_k^m \cdot \boldsymbol{w}_k^n}{\sqrt{\boldsymbol{w}_k^m \cdot \boldsymbol{w}_k^m}\sqrt{\boldsymbol{w}_k^n \cdot \boldsymbol{w}_k^n}} = \frac{R_k^{m,n}}{\sqrt{Q_k^m\, Q_k^n}}\,. \tag{7}$$

The maximum value $\rho_k = 1$ is reached for fully synchronized hidden units (zero distance), while uncorrelated weight vectors have zero overlap (maximal distance).

The distance between two corresponding hidden units decreases, if the learning rule (4) is applied to both of them using the same input vector $\boldsymbol{x}_k$. Thus coordinated moves of the weights have an attractive effect in the process of synchronization [6,15].

But changing the weights in only one hidden unit increases the average distance [6]. These repulsive steps between two corresponding hidden units can only occur if their output bits are different [16]. The probability for this event is given by the well-known generalization error of the perceptron [13]

$$\epsilon_k = \frac{1}{\pi} \arccos \rho_k \,, \tag{8}$$

which depends on the overlap ρ_k between the weight vectors of corresponding hidden units. For an attacker using the same learning rule as A and B, repulsive steps in the kth hidden unit occur with probability $p_\mathrm{r} = \epsilon_k$, as E cannot influence the process of synchronization.

In contrast, the partners update the weights in their TPMs only if $\tau^\mathrm{A} = \tau^\mathrm{B}$. In the case of identical generalization error, $\epsilon_k = \epsilon$, we find for $K = 3$ that repulsive steps in a hidden unit of B's neural network occur with the probability [16]

$$p_\mathrm{r} = \frac{2(1-\epsilon)\epsilon^2}{(1-\epsilon)^3 + 3(1-\epsilon)\epsilon^2} \leq \epsilon \,. \tag{9}$$

So p_r is lower for synchronization than for learning and the partners partially avoid repulsive steps. This advantage makes neural cryptography feasible and prevents successful attacks, which are only based on simple learning.

But E can assign a confidence level to each output σ_k^E of its hidden units. For this task the local field

$$h_k = \frac{1}{\sqrt{N}} \boldsymbol{w}_k \cdot \boldsymbol{x}_k \,, \tag{10}$$

is used as additional information. Then the prediction error, the probability of different output bits for an input vector $\boldsymbol{x}_k$ inducing a local field h_k, is given by [17]

$$\epsilon(\rho_k, h_k) = \frac{1}{2} \left[1 - \mathrm{erf} \left(\frac{\rho_k}{\sqrt{2(1-\rho_k^2)}} \frac{|h_k|}{\sqrt{Q_k}} \right) \right] . \tag{11}$$

The function $\epsilon(\rho_k, h_k)$ reaches a maximum of $\epsilon(\rho_k, 0) = 0.5$ for $h_k = 0$. In this case $\boldsymbol{x}_k$ is perpendicular to $\boldsymbol{w}_k$ and the neural network has no information about this example. But for increasing $|h_k|$ the confidence of prediction rises and $\epsilon(\rho_k, h_k)$ shrinks.

This effect is essential for the *Geometric Attack* [6], which is—up to now—the most successful method for a single attacking neural network with a structure identical to A's and B's [9]. Here E trains its own TPM, with the examples, input vectors and output bits, transmitted by the two partners. If $\tau^\mathrm{E} = \tau^\mathrm{A}$, the attacker applies the same learning rule as A and B. But for $\tau^\mathrm{E} \neq \tau^\mathrm{A}$ E knows that at least one of the hidden units has made a wrong prediction and searches for the unit k with the minimum absolute value $|h_k|$ of the local field. Because this hidden unit has the lowest confidence of prediction, its output σ_k^E is most likely to be different from σ_k^A. Therefore the attacker inverts both σ_k^E and τ^E. Afterwards the usual learning rule can be applied. As this geometric attack method reduces the frequency of repulsive steps, E increases its success probability P_E by taking the local field into account.

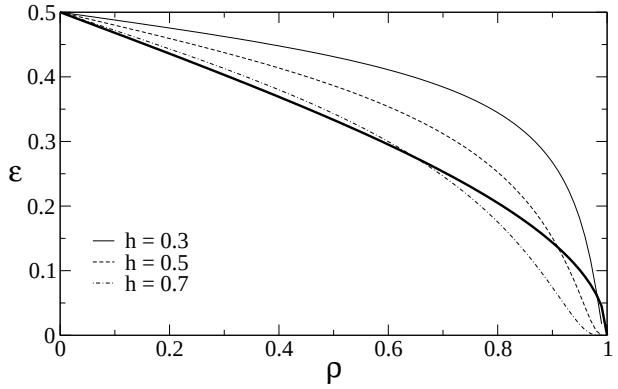

Figure 2. Prediction error $\epsilon(\rho, h)$ as a function of the overlap ρ for different values of the local field. The generalization error ϵ is shown as thick line. We assumed $Q = 1$ for this diagram.

But the partners can influence the local field. Instead of using random inputs, they are able to select input vectors with a fixed $|h_k|$ (queries [12]) in their own hidden units. This essentially modifies the functional dependency between overlap ρ_k and frequency of repulsive steps p_r. In this case (11) determines the probability of different output bits instead of (8).

Note that the chosen absolute local field $|h_k|$ for synchronization with queries is lower than the average value

$$\langle |h_k| \rangle = \sqrt{2Q_k/\pi} \approx 0.8\sqrt{Q_k} \tag{12}$$

observed for random inputs. Therefore the overlap

$$\rho_{k,\mathrm{in}} = \frac{\boldsymbol{w}_k \cdot \boldsymbol{x}_k}{\sqrt{\boldsymbol{w}_k \cdot \boldsymbol{w}_k}\sqrt{\boldsymbol{x}_k \cdot \boldsymbol{x}_k}} = \frac{1}{\sqrt{N}} \frac{h_k}{\sqrt{Q_k}} \tag{13}$$

between input vector and weight vector converges to zero in the limit $N \to \infty$, even if queries with $0 < |h_k| < \infty$ are used.

Figure 2 shows the prediction error $\epsilon(\rho, h)$ for queries, which induce the local field h. Compared to the case of random inputs, $\epsilon(\rho, h)$ is increased for small overlap and decreased for nearly synchronized neural networks. By selecting queries with different local fields, A and B can regulate this effect.

As learning is slower than synchronization, ρ^{AE} is typically smaller than ρ^{AB}. In this situation queries with small values of the parameter h increase the frequency of repulsive steps for the attacker without affecting the process of synchronization too much.

4. Synchronization

In order to integrate queries into the neural key-exchange protocol [1], only the generation of the inputs has to be changed. Instead of choosing completely random $x_{k,j}$, both partners alternately ask each other questions. In each time step either A or B uses the algorithm described in Appendix A to generate K input vectors $\boldsymbol{x}_k$, which result in $h_k^{\mathrm{A/B}} \approx \pm H$. Then these queries are sent to the other partner and both calculate the output of their TPMs. After the exchange of τ^{A} and τ^{B}, the Hebbian

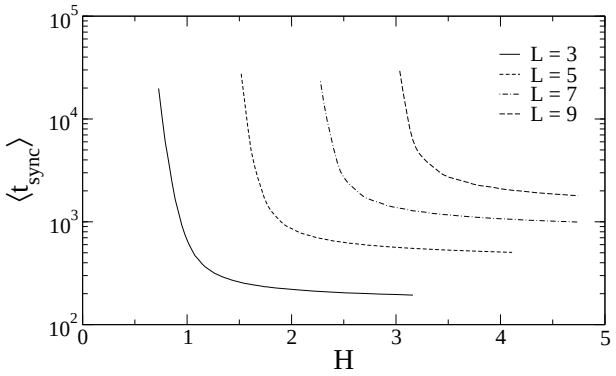

Figure 3. Synchronization time of two TPMs with $K = 3$ and $N = 1000$, averaged over 10 000 simulations.

learning rule is used to update the weights as described in section 2. This leads to synchronization after t_{sync} steps.

As shown in figure 3, $\langle t_{\mathrm{sync}} \rangle$ diverges for $H \to 0$. In this limit the prediction error $\epsilon(\rho_k, H)$ reaches 0.5 independent of ρ_k. Therefore the effect of the repulsive steps inhibits synchronization. But the choice of H does not influence $\langle t_{\mathrm{sync}} \rangle$ much, as long as this quantity is large enough.

The dependence on L of $\langle t_{\mathrm{sync}} \rangle$ is caused by two effects:

- The dynamics of each weight can be described as random walk with reflecting boundaries if the control signals σ_k and τ are neglected. Calculations for this simplified model of neural synchronization show that $\langle t_{\mathrm{sync}} \rangle$ scales proportional to L^2 [15]. This behavior has been observed in neural cryptography with random inputs [9], too.
- If queries are used, the probability of repulsive steps p_{r} depends not only on the overlap ρ_k, but also on the quantity $|h_k|/\sqrt{Q_k}$. Assuming uniformly distributed weights, we find

$$\langle Q_k \rangle = \left\langle \frac{1}{N} \boldsymbol{w}_k \cdot \boldsymbol{w}_k \right\rangle = \frac{1}{3} L(L+1) \sim \frac{1}{3} L^2 \tag{14}$$

  for the expectation value of the order parameter Q_k. Hence we have to increase H proportional to the synaptic depth L, if we want to observe the same frequency of repulsive steps.

Using both $\langle t_{\mathrm{sync}} \rangle \propto L^2$ and $H \propto L$ we can rescale $\langle t_{\mathrm{sync}} \rangle_{H,L}$ in order to obtain functions $f_L(\alpha)$, which are nearly independent of the synaptic depth in the case $L \gg 1$:

$$\langle t_{\mathrm{sync}} \rangle = L^2 f_L\left(\frac{H}{L}\right) . \tag{15}$$

In figure 4 we have plotted these functions for different values of L. It is clearly visible that $f_L(\alpha)$ converges to a universal scaling function $f(\alpha)$ in the limit $L \to \infty$:

$$f(\alpha) = \lim_{L \to \infty} f_L(\alpha) . \tag{16}$$

Additionally, we find that the distance $|f_L(\alpha) - f(\alpha)|$ shrinks proportional to L^{-1}. Therefore we can use finite size scaling to determine $f(\alpha)$, which is shown in figure 4, too.

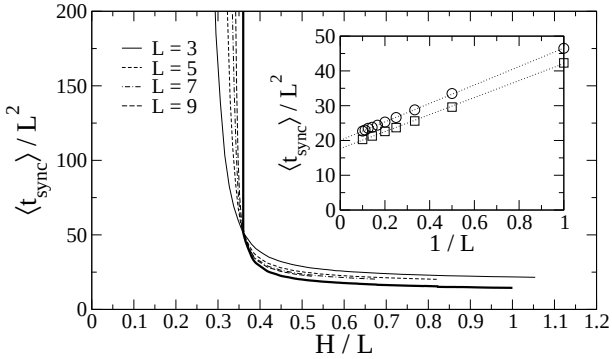

Figure 4. Scaling behavior of the synchronization time. The thick curve denotes the universal function $f(\alpha)$ defined in (16). It has been obtained by finite size scaling, which is shown in the inset for $\alpha = 0.5$ (○) and $\alpha = 0.6$ (□).

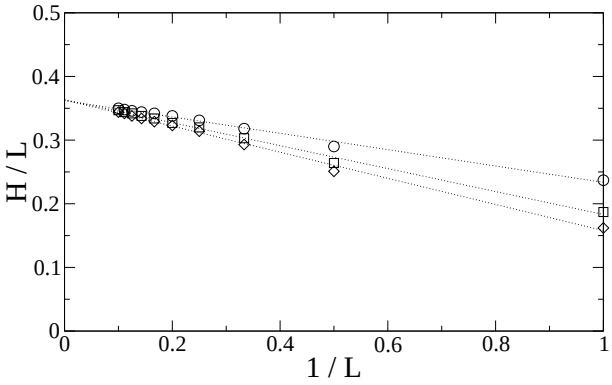

Figure 5. Extrapolation of f_L^{-1} to $L \to \infty$. Symbols denote the values extracted from figure 4 for the average synchronization times $100L^2$ (○), $150L^2$ (□) and $200L^2$ (◇).

This function diverges for $\alpha < \alpha_c$. We estimate $\alpha_c \approx 0.36$ for $K = 3$ and $N = 1000$ by extrapolating the inverse function f_L^{-1} as shown in figure 5. Consequently, synchronization is only achievable for $H > \alpha_c L$ in the limit $L \to \infty$.

5. Security

Up to now, the most successful attack on neural cryptography is the *Majority Flipping Attack* [11], which is an extension of the *Geometric Attack*. Instead of a single neural network, E uses an ensemble of M TPMs. At the beginning, the weight vectors of all attacking networks are chosen randomly, so that the average initial overlap between them is zero. Like A and B, the attacker only updates the weights in time steps with $\tau^A = \tau^B$. If the output $\tau^{E,m}$ of the mth attacking network disagrees with τ^A, the hidden unit with the smallest absolute value $|h_k^{E,m}|$ of the local field is selected. Then the output bits $\sigma_k^{E,m}$ and $\tau^{E,m}$ are inverted. Afterwards E counts the internal representations $(\sigma_1^{E,m}, \ldots, \sigma_K^{E,m})$ and selects the most common one. This majority vote is then adopted by all attacking networks for the application of the learning rule.

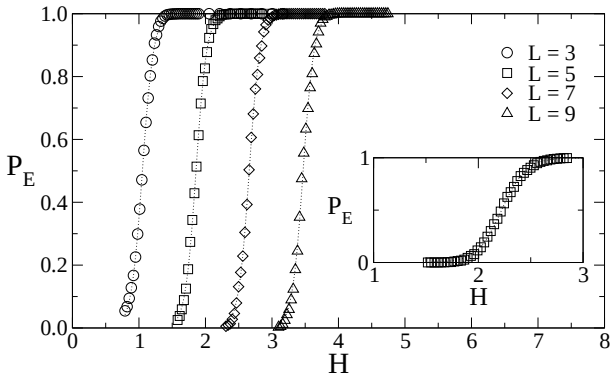

Figure 6. Success probability of the *Majority Flipping Attack* as a function of H. Symbols denote the results obtained in 10 000 simulations for $K = 3$, $M = 100$ and $N = 1000$. The inset shows the success of a *Geometric Attack* for $K = 3$, $L = 5$ and $N = 1000$.

Because of these identical updates, E's neural networks become correlated [11], which reduces the efficiency of the *Majority Flipping Attack*. In order to slow down this effect, we use the modifications proposed in [11]:

- The majority vote is only considered in even time steps. Otherwise E updates the weights according to the internal representation of the particular neural network.
- At the beginning of the synchronization, only the *Geometric Attack* is applied by the attacker. But instead of waiting until $t > \frac{1}{3}\, t_{\mathrm{sync}}$ as suggested in [11], E starts with the *Majority Flipping Attack* after 100 steps.

For the neural key-exchange protocol with random inputs and Hebbian learning rule, the success probability P_{E} of this method reaches a constant non-vanishing value in the limit $L \to \infty$ [11]. Therefore it breaks the security of this type of neural cryptography.

In contrast, if the two partners A and B use queries for the neural key exchange, the success probability strongly depends on the parameter H. This can be used to regain security against the *Majority Flipping Attack*. As shown in figure 6 a Fermi-Dirac distribution

$$P_{\mathrm{E}} = \frac{1}{1 + \exp(-\beta(H - \mu))} \tag{17}$$

with two parameters β and μ is a suitable fitting function for describing P_{E} as a function of H. Equation (17) is also valid for the *Geometric Attack*, which is a special case of the *Majority Flipping Attack* ($M = 1$).

Figure 7 shows the dependence on L of the fit parameter μ for both attacks. Aside from finite size effects for small values of L, this parameter is proportional to the synaptic depth of the TPMs:

$$\mu = \alpha_{\mathrm{s}}\, L\,. \tag{18}$$

Obviously, the quantity $\alpha = H/L$ not only determines the synchronization time but also the success of an attack. These effects are caused by the modification of p_{r} due to the use of queries. The other parameter β is nearly constant for $L > 3$. So both α_{s} and β are independent of the chosen parameters H and L, but depends on the attacker's method.

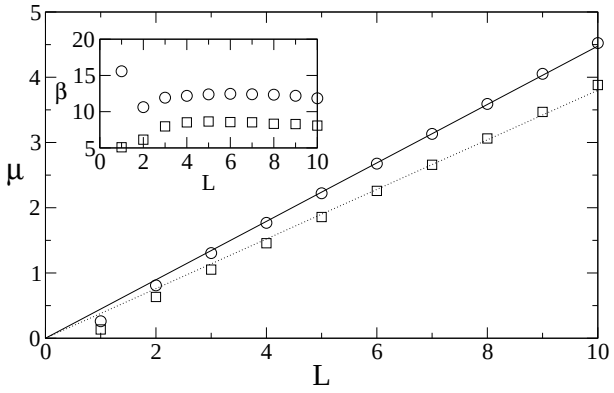

Figure 7. Parameters μ and β as a function of the synaptic depth L. Symbols denote the results of fits using (17) for the *Geometric Attack* ($\circ$) and the *Majority Flipping Attack* with $M = 100$ ($\square$). From these values we obtain $\alpha_{\mathrm{s,flip}} \approx 0.45$ (——) and $\alpha_{\mathrm{s},M=100} \approx 0.38$ ($\cdots\cdots$) according to (18).

From these results we can also deduce the scaling behavior of P_{E} as a function of the synaptic depth. For both the *Majority Flipping Attack* and the *Geometric Attack* we obtain

$$P_{\mathrm{E}} = \frac{1}{1 + \exp(\beta(\alpha_{\mathrm{s}} - \alpha)L)} \tag{19}$$

for synchronization with queries and $H = \alpha L$. In the limit $L \to \infty$, the asymptotic behavior of P_{E} is given by

$$P_{\mathrm{E}} \sim \mathrm{e}^{-\beta(\alpha_{\mathrm{s}} - \alpha)L} \tag{20}$$

as long as $\alpha < \alpha_{\mathrm{s}}$. Here the success probability P_{E} decreases exponentially with increasing synaptic depth,

$$P_{\mathrm{E}} \propto \mathrm{e}^{-yL}\,, \tag{21}$$

which is also observed in the case of random inputs, if E uses the *Geometric Attack* [9].

Consequently, the neural key exchange with queries is secure against both attacks. An attacker using the *Majority Flipping Attack* only decreases the value of $y = \beta(\alpha_{\mathrm{s}} - \alpha)$, because $\alpha_{\mathrm{s,majority}} < \alpha_{\mathrm{s,flip}}$, but does not change the exponential scaling law (21).

For practical aspects of security, however, one has to look at the function $P_{\mathrm{E}}(\langle t_{\mathrm{sync}} \rangle)$, which is shown in figure 8. Here we find that queries enhance the security of the neural key-exchange protocol a lot for given synchronization time. There is an optimal value of H associated with each L, which lies on the envelope of all functions $P_{\mathrm{E}}(\langle t_{\mathrm{sync}} \rangle)$.

Figure 8 also shows that A and B can achieve higher security against attacks than predicted by (19) with $\alpha = \alpha_{\mathrm{c}}$ as long as L is not to large. This phenomenon is caused by finite size effects which enable synchronization even for $H < \alpha_{\mathrm{c}} L$. But, of course, this does not work for $L \gg 1$. Therefore the envelope and $P_{\mathrm{E}}(\langle t_{\mathrm{sync}} \rangle)$ for $H = \alpha_{\mathrm{c}} L$ converge asymptotically in the limit $L \to \infty$.

Queries reveal additional information about the weight vectors of A's and B's neural networks. However, an attacker E cannot benefit from this information, since for a given value of H, there is still an exponential large number of weight vectors $\boldsymbol{w}_k$,

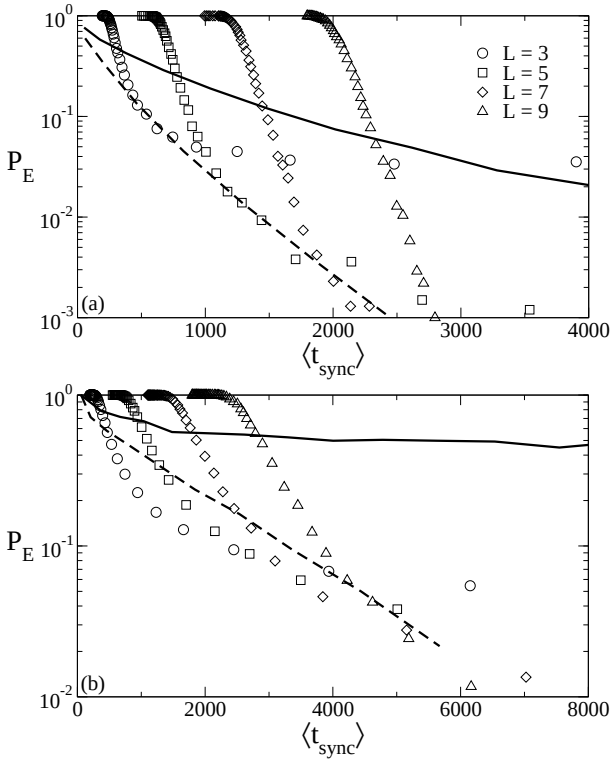

Figure 8. Success probability as a function of the average synchronization time for $K = 3$, $N = 1000$ and different values of L and H. Part (a) shows the result for the *Geometric Attack* and part (b) for the *Majority Flipping Attack* with $M = 100$ attacking neural networks. The solid curve in each graph represents the success probability for neural cryptography with random inputs and the dashed line marks $H = \alpha_c L$.

which are consistent with a given query. As an example, there are 2.8×10^{129} possible weight vectors for $L = 10$, $N = 100$, and $h_k = 10$. Because of this large number, E cannot gain useful information from queries.

6. Conclusions

Neural cryptography is a subtle competition between interaction and learning of neural networks. Two neural networks A and B exchange some information about their states over a public channel. The amount of information has to be so high that the two networks A and B can synchronize. But it has to be so low that an attacker E can only synchronize with a low probability which can be decreased to an arbitrary low value.

We have shown that increasing the amount of exchanged information can be of advantage for cryptography. We have included queries in the training process of the neural networks. This means that alternately A and B are generating an input which is correlated with its state and A or B is asking the partner for the corresponding output bit. The overlap between input and weight vector is so low that the additional information does not reveal much about the internal states. But queries introduce a

mutual influence between A and B which is not available to an attacking network E. In addition the method obtains a new (public) parameter which can be adjusted to give optimal security.

We have applied this new method to the case of the Hebbian training rule which was successfully attacked using the majority of an ensemble of attackers. We find that queries restore the security of the method: the probability of a successful naive majority attack can be decreased to any desired level.

Appendix A. Generation of queries

In this appendix we describe the algorithm used to generate a query $\boldsymbol{x}_k$, which induces a previously chosen local field h_k. The solution, of course, depends on the weight vector $\boldsymbol{w}_k$ of the hidden unit. This task is similar to solving a knapsack problem [18], which can be very difficult. But we need only a fast approximate solution in order to use queries in the neural key-exchange protocol.

As both weights $w_{k,j}$ and inputs $x_{k,j}$ are discrete, there are only $2L+1$ possibilities for $w_{k,j} \cdot x_{k,j}$. Therefore we can describe the solution by counting the number $c_{k,l}$ of products with $w_{k,j} \cdot x_{k,j} = l$. Then the local field is given by:

$$h_k = \frac{1}{\sqrt{N}} \sum_{l=1}^{L} l(c_{k,l} - c_{k,-l}) \,. \tag{A.1}$$

We also note that the sum $n_{k,l} = c_{k,l} + c_{k,-l}$ is equal to the number of weights with $|w_{k,j}| = |l|$. Hence the values of $n_{k,l}$ depend only on the weight vector $\boldsymbol{w}_k$. This can be used to write h_k as a function of only L variables, because the generation of queries cannot change $\boldsymbol{w}_k$:

$$h_k = \frac{1}{\sqrt{N}} \sum_{l=1}^{L} l(2c_{k,l} - n_{k,l}) \,. \tag{A.2}$$

In our simulations we use the following algorithm to generate the queries. First the output σ_k of the hidden unit is chosen randomly. Therefore the set value of the local field is given by $h_k = \sigma_k H$. Then we use either

$$c_{k,l} = \left\lfloor \frac{n_{k,l}+1}{2} + \frac{1}{2l} \left(\sigma_k H \sqrt{N} - \sum_{j=l+1}^{L} j(2c_{k,j} - n_{k,j}) \right) \right\rfloor \tag{A.3}$$

or

$$c_{k,l} = \left\lceil \frac{n_{k,l}-1}{2} + \frac{1}{2l} \left(\sigma_k H \sqrt{N} - \sum_{j=l+1}^{L} j(2c_{k,j} - n_{k,j}) \right) \right\rceil \tag{A.4}$$

to compute the values of $c_{k,L}$, $c_{k,L-1}$, ..., $c_{k,1}$. In each calculation one of the two equations is selected randomly with equal probability, so that rounding errors do not influence the average result. Additionally, we have to take into account that $0 \le c_{k,l} \le n_{k,l}$. Therefore we set $c_{k,l}$ to the nearest boundary value, if (A.3) or (A.4) yield a result outside this range.

Afterwards the input vector $\boldsymbol{x}_k$ is generated. Inputs associated with zero weights are chosen randomly, because they do not influence the local field. The other input bits $x_{k,j}$ are divided into L groups according to the absolute value $l = |w_{k,j}|$ of their

corresponding weight. In each group, $c_{k,l}$ inputs are selected randomly and set to $x_{k,j} = \mathrm{sgn}(w_{kj})$. The remaining $n_{k,l} - c_{k,l}$ input bits are set to $x_{k,j} = -\mathrm{sgn}(w_{k,j})$.

Simulations show that the absolute local field $|h_k|$ matches its set value H on average. And we observe only small deviations, which are caused by the restriction of the input values to $+1$ or -1. So we can generate queries, which approximately induce a predetermined absolute local field H by using this algorithm.

References

[1] Kanter I, Kinzel W and Kanter E, *Secure exchange of information by synchronization of neural networks*, 2002 *Europhys. Lett.* **57** 141 [cond-mat/0202112]

[2] Kinzel W and Kanter I, *Disorder generated by interacting neural networks: application to econophysics and cryptography*, 2003 *J. Phys. A: Math. Gen.* **36** 11173

[3] Stinson D R, 1995 *Cryptography: Theory and Practice* (Boca Rotan, FL: CRC Press)

[4] Metzler R, Kinzel W and Kanter I, *Interacting neural networks*, 2000 *Phys. Rev.* E **62** 2555 [cond-mat/0003051]

[5] Kinzel W, Metzler R and Kanter I, *Dynamics of interacting neural networks*, 2000 *J. Phys. A: Math. Gen.* **33** L141 [cond-mat/9906058]

[6] Klimov A, Mityaguine A and Shamir A, *Analysis of Neural Cryptography*, 2003 *Advances in Cryptology—ASIACRYPT 2002*, ed Y Zheng (Heidelberg: Springer) p 288

[7] Kinzel W and Kanter I, *Interacting neural networks and cryptography*, 2002 *Advances in Solid State Physics*, ed B Kramer (Berlin: Springer) vol 42 pp 383–391 [cond-mat/0203011]

[8] Kinzel W and Kanter I, *Neural Cryptography*, 2002 *Preprint* cond-mat/0208453

[9] Mislovaty R, Perchenok Y, Kanter I and Kinzel W, *Secure key-exchange protocol with an absence of injective functions*, 2002 *Phys. Rev.* E **66** 066102 [cond-mat/0206213]

[10] Kanter I and Kinzel W, *The theory of neural networks and cryptography*, 2003 *Proceedings of the XXII Solvay Conference on Physics on the Physics of Communication*, ed I Antoniou, V A Sadovnichy and H Wather (Singapore: World Scientific) p 631

[11] Shacham L N, Klein E, Mislovaty R, Kanter I and Kinzel W, *Cooperating attackers in neural cryptography*, 2004 *Phys. Rev.* E **69** 066137 [cond-mat/0312068]

[12] Kinzel W and Rujan P, *Improving a network generalization ability by selecting examples*, 1990 *Europhys. Lett.* **13** 473

[13] Engel A and Van den Broeck C, 2001 *Statistical Mechanics of Learning* (Cambridge: Cambridge University Press)

[14] Kinzel W, *Theory of Interacting Neural Networks*, 2002 *Preprint* cond-mat/0204054

[15] Ruttor A, Reents G and Kinzel W, *Synchronization of random walks with reflecting boundaries*, 2004 *J. Phys. A: Math. Gen.* **37** 8609 [cond-mat/0405369]

[16] Ruttor A, Kinzel W, Shacham L and Kanter I, *Neural cryptography with feedback*, 2004 *Phys. Rev.* E **69** 046110 [cond-mat/0311607]

[17] Ein-Dor L and Kanter I, *Confidence in prediction by neural networks*, 1999 *Phys. Rev.* E **60** 799

[18] Schroeder M R, 1986 *Number Theory in Science and Communication* (Berlin: Springer) 2nd edn